2 COMP15111 Lab 2 – Control Structures

Duration: 1 lab session

2.1 Aims

To practise converting simple Java statements into ARM assembly code.

2.2 Learning Outcomes

On successful completion of this exercise a student will:
– have encoded Java assignment, If and While statements using ARM instructions
– have used various techniques to improve the resulting ARM code

2.3 Summary

Translate 3 examples of Java control structures into ARM code and then improve the resulting code in various ways. (The examples illustrate the implementation and use of simple input and output and are based on various programs used in the Java course-unit.)

Each lab exercise has the usual deadline at the end of your scheduled lab session. If you attend the lab you will, if you need it, get an automatic 7-day extension (up to the usual starting time of the lab). This is so that you can get help in the lab and then have some time to make use of that help to complete the exercise. This will be the case for all your labs for this course-unit. Remember that you must use “submit” in the usual way to show that you completed your work by the (extended) deadline. If you decide to complete the exercise after our labs shut in the evening, it is your responsibility to make sure that you can “submit” remotely.

2.4 Description

As usual, you will need to copy the starting code to your lab directory and run KMD:

```bash
> cd COMP15111/ex2
> cp /opt/info/courses/COMP15111/Lab/ex2/* .
> start_komodo 15111&
```

In all 5 parts, the programs you run will attempt to perform output and possibly some input as well. To view the output, you will need to open a “Features” window – go to the “Special” menu at the top left of the KMD window and select the “Features” menu item. You should see a window like that illustrated to the right:

You probably ought to use the “Clear” button each time before you run a program.

2.4.1 Part 1 – Hello Someone version 1

Your copy of the file “part1.s” contains the skeleton of an ARM program, with some (approximately) Java code in comments. Currently, the program outputs "Hello " in the Features window, waits for you to type any character into the Features window, and then outputs "and good-bye!" and stops.

**TASK:** Edit the program (e.g. using nedit) to insert ARM instructions that are a translation of the Java code in the comments. (Where a comment refers to R0, you should just use register R0, rather than a variable.) Do not change any of the ARM instructions already in the program. (The “return” character, that you use to end your input and that you need to match in the program, is encoded by 10 in the ASCII character set that we are using).

The rules on text case vary according to the assembler in use. In the labs the assembler is *case-insensitive* with respect to mnemonics and register names but *case-sensitive* with respect to labels. Thus “R5” and “r5” are the same, as are “ADD”, “add” or even “Add”; however “jack”, “JACK”, “Jack” etc. are all different.
Compile, load, reset and run your edited code; remember to check the log pane to see that there are no syntax errors. If you get it right, after outputting “Hello” your program will wait for you to type your name into the Features window. As you type in your name, it will echo the characters, so you see e.g. “Hello Javier”. When you have finished typing in your name, press the return key and the program will then output “and good-bye!” on the next line.

2.4.2 Part 2 – Hello Someone version 2

Your copy of the file “part2.s” contains the skeleton of an ARM program, with some (approximately) Java code in comments. The program is very similar to part1, but tries to output everything on one line. To do this, it has to avoid echoing the return you use to indicate the end of your name.

**TASK:** Edit the program to insert some ARM instructions that are a translation of the Java code in the comments. (As before, where this refers to R0, you should just use register R0, rather than a variable.) Do not change any of the ARM instructions already in the program.

Compile, load, reset and run your edited code. If you get it right, the program will behave as in part 1, except that all the output will be on one line e.g. “Hello Pete and good-bye!”.

2.4.3 Part 3 – Improving the code for part 1

**TASK:** Copy your answer in “part1.s” to a file “part3.s” which you will edit to improve the code: `cp part1.s part3.s`  
Optimize your code by reducing the number of branch instructions that need to be executed in each iteration.

Why is your modified program more efficient?

2.4.4 Part 4 – Age History

Your copy of the file “part4.s” contains the skeleton of an ARM program, with some (approximately) Java code in comments. Currently, the program just contains instructions to generate the output, assuming that the variables were set correctly.

**TASK:** Edit the program to insert ARM instructions that are a translation of the Java code in the comment. Do not change any of the ARM instructions already in the program. Compile, load, reset and run your edited code.

If you get it right, you should get essentially the same output as for the version of the program given in the Java course-unit.

2.4.5 Part 5 – Improving the code for Age History

**TASK:** Copy your answer in “part4.s” to a file “part5.s” which you will edit to improve the code: `cp part4.s part5.s`  
You should change your code, and the code you were given, to keep the variables “present”, “birth”, “year” and “age” in registers (R4, R5, R6 and R7 respectively) at all times and avoid storing them back to memory.

You will need to add instructions to initialise these variables to the correct values at the start of the program. You will also need to modify several instructions that load values into R0 before “SVC” is used to output them. If you do this correctly, when you run your program you should get the same output as before.

2.5 Completion, Feedback and Marking Process

There are 2 marks for each part. (You cannot get different parts marked on different days.)

As soon as you have completed the exercise, you need to run `submit` (making sure you are in the correct directory – COMP15111/ex2 for this exercise). `submit` records the time that you completed the exercise and archives your work.

After this, run `labprint` to generate a marking sheet (again, make sure you are in COMP15111/ex2, type the command `labprint` in a shell window and collect your marking sheet from a printer) and then ask a demonstrator to mark your work (you may need to add your name and seat number to a list of those queueing to be marked). You will be asked the question at the end of part 3.

**IMPORTANT** – University rules say that all work must be marked within 15 days, so you must get your work marked during your next scheduled lab (unless have a good excuse e.g. that you were ill). If a lab does not get marked by this deadline, it may get removed from Arcade and, hence, will not count towards the final mark. This will be the case for all of your labs for this course-unit.